FIR Filter Implementation: A Hilbert Transformer
ECE 443/643 Communications Lab #7: NDSU : Spring 2008

Roger Green, Brian Chapman, David Farden

Abstract

In this lab, we will construct a digital finite impulse response (FIR) filter using Code Composer Studio (CCS) and the TMS320C6713 DSP from Texas Instruments. FIR filters are quite popular since they (1) are easy to implement, (2) are always stable, and (3) are flexible enough to implement a wide variety of desired system functions. For this lab, we will concentrate on the implementation of a digital system that functions, at least approximately, as a Hilbert transformer. The "digital wire" provided in Lab # 6 will provide a starting point for the system design.

1 Prelab, Week 1

Be sure to read the entire laboratory handout prior to attending lab. That's it! Pay particular attention to understanding what you need to accomplish in the first week of the lab – the design and implementation of a generic FIR filter:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k].$$

2 Implement an FIR Filter (Week 1, possibly part of Week 2)

A causal digital LTI filter can be characterized by its impulse response: $h[n]$. If the impulse response has finite duration (i.e., it is non-zero only on some interval $0 \leq n \leq N-1$), then the filter is called a finite impulse response (FIR) filter.

A length-N digital FIR filter is characterized by a convolutional sum:

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k],$$

where $x[n]$ is the input at discrete-time n and $y[n]$ is the output at discrete-time n. Notice, the output at any time only depends on the current and $N-1$ past input values weighted by values from the impulse response. Further, notice that this operation is essentially the inner (dot) product between a length-N input vector and the length-N impulse response.

1. Verify that the impulse response of this system is $h[n]$.

Create a program to implement an arbitrary order FIR filter on the TMS320C6713 DSP. Your code should roughly operate as follows:

- Main program should perform all initializations and then enter a do-nothing loop as it waits for an interrupt (indicating a sample is available and that an output sample can be generated). Be sure to set the sampling rate of the system to some reasonable value, $F_s \geq 20$kHz.
- In the interrupt routine, the new input sample should be collected, cast to a "float" data type, and placed into the proper location of an N-length input vector. Two possible ways to implement the input vector are: (1) always put the new sample at the "top", which requires the entire vector to be "shifted down" before the interrupt routine is ended to ensure the top space is available for the next input, or (2) use a circular buffer where the new input sample is simply written over the oldest sample in the buffer. An advantage of (1) is that indexing the vector is very simple – the first index is always the most

recent sample while the last (Nth) index is always the oldest sample. A significant disadvantage of (1) is that there is a lot of overhead to shift the vector elements every interrupt, particularly for large N. An advantage of (2) is that there is no shifting overhead; the accompanying disadvantage of (2) is that more sophisticated indexing is required. For maximum credit, you should implement (using separate projects) both structures.

Circular indexing (or more properly, circular addressing) can be accomplished using a modulo-type operator. If you have never used a modulo-operator, your T.A. will acquaint you with the idea. Although C provides a modulo operator (%), it is **extremely** inefficient. Thus, it is recommended that you use a masking operation using a logical AND (&). This will likely require that your FIR filter length N be a power of two. Again, your T.A. can acquaint you with these ideas if you have never seen them before.

- Be sure that the length-N impulse response vector is available. It is recommended that you put the coefficients in a header file and then include that header file in your main code. In this way, you can easily change the coefficients of your filter (just change the header file), thereby changing the behavior of the system. For the time being, use $N = 30$ and define $h[n] = \frac{1}{N}(u[n] - u[n - N])$.

- Compute the current output sample by (essentially) taking the inner (dot) product of the length-N input vector and the length-N impulse response vector. Notice that your computation of the desired inner product will depend on your structure for your input vector. Circular indexing/addressing will required if your input is a circular buffer.

  Once the output value is computed (presumably as a float), cast the value to the proper data type for output, send it to the DAC, and exit the interrupt routine.

Once your program is complete, verify its correct operation. If you are inefficient in your programming, you may fail to meet timing (i.e., your interrupt service routine takes more than $1/F_s$ seconds). Your T.A. can assist you to configure CCS to compile more efficient code. I have seen cases where using compiler optimizations allow nearly an order-in-magnitude increase in performance.

2. Determine and plot the magnitude response for the test filter $h[n] = \frac{1}{N}(u[n] - u[N])$. Your frequency response plot should use digital frequencies Ω ($-\pi \leq \Omega \leq \pi$) as the independent variable. Recall, $\Omega = \pi$ corresponds to the folding frequency $F_s/2$.

3. What type of filter is $h[n]$ (LP, HP, BP, BS, or none)? If appropriate, identify the filter's 3-dB point(s) in Hertz given the chosen sampling rate of your system.

4. How long does it take an FIR filter to reach a steady-state response if the input is a sinusoid? If possible to determine, how long does it take to reach steady-state if the input is non-sinusoidal?

5. How were you able to test your system for correct operation?

3 Prelab, Week 2

Determine the coefficients $h[n]$ ($0 \leq n \leq N - 1$) so that the system operates as a Hilbert Transformer. To help you in this task, consider the following.

The ideal frequency response of a (non-causal) digital Hilbert transformer is ($\Omega = 2\pi fT$, $T = 1/F_s$, $|\Omega| \leq \pi$):

$$H(\Omega) = \begin{cases} -j & \Omega > 0 \\ 0 & \Omega = 0 \\ j & \Omega < 0 \end{cases}$$

By shifting the impulse response to the right by $(N - 1)/2$ and then truncating the response to include the $(0 \leq n \leq N - 1)$ terms, a causal N-length approximation is possible.

Correspondingly, the ideal response including the $(N-1)/2$ shift is:

$$H_{\text{shift}}(\Omega) = \begin{cases} -je^{-j\Omega(N-1)/2} & \Omega > 0 \\ 0 & \Omega = 0 \\ je^{-j\Omega(N-1)/2} & \Omega < 0 \end{cases}$$

The non-truncated but shifted impulse response can thus be obtained by taking the inverse DTFT of $H_{\text{shift}}(\Omega)$.

- Determine an expression for $h_{\text{shift},N}[n]$, the impulse response of non-truncated but shifted discrete Hilbert Transformer. The subscript N is included to simply emphasize that the impulse response is a function of the desired length of the end filter (and thus the shift applied).
- Write a MATLAB program that will: (1) evaluate the expression $h_{\text{shift},N}[n]$ for arbitrary N over the truncated range $(0 \leq n \leq N-1)$, (2) automatically generates a header file with these coefficient values for inclusion in your C-program, and (3) plots the impulse response function $h_{\text{shift},N}[n]$ as well as the corresponding magnitude response and phase response plots. For part (2), it will be helpful to learn about the `fprintf` command in MATLAB, which functions similarly in C.
- Using the code you developed, design two digital Hilbert transformers: one for $N = 29$ and one for $N = 30$. Do the results seem reasonable? Do you expect one filter to behave better than the other? Why? Try to predict how well these filters will operate.

4 Implement an Digital Hilbert Transformer (Week 2)

Complete your FIR filter implementation from the first week if not done already. Notice that the Hilbert transformers designed in the pre-lab are implemented as digital FIR filters.

- Using the results for your pre-lab, implement your $N = 29$ digital Hilbert transform filter using your FIR filter code. Determine a method to test the functionality of the filter, taking care to fully document functionality testing in your lab report.
- Using the results for your pre-lab, implement your $N = 30$ digital Hilbert transform filter using your FIR filter code. Determine a method to test the functionality of the filter, taking care to fully document functionality testing in your lab report.
- Which, if either, filter behaves best? Justify your answer.
- Use the code you developed in the pre-lab to design a new Hilbert transformer. Your goal is to implement the highest order filter possible while still meeting timing constraints on the DSP. What order were you able to obtain? Do the results appear better than the previous implementations? How can you be sure you are meeting timing? If you are really lost on this part, consult your T.A. for strategies (e.g., LED toggle in ISR, all-pass test filters, etc.).

2. Discuss the general modifications needed to adapt your code to implement a single-side band digital modulation system. Who knows, you might be asked to actually do this in an upcoming lab...

5 Reporting Requirements

To receive credit for this laboratory, you need to do a few things:

1. Complete all steps discussed in the laboratory handout.
2. Summarize your approach and findings in a concise, professional report. Be sure to answer all questions posed in the handout.
3. Attach a copy of your C-code programs as an appendix to your report.